ТЕХНИЧЕСКИ УНИВЕРСИТЕТ–СОФИЯ

КАТЕДРА “КОМПЮТЪРНИ СИСТЕМИ”

*„ПАРАЛЕЛНО ПРОГРАМИРАНЕ“*

***УПРАЖНЕНИЕ 2***

---

# *СОФТУЕРЕН ПАКЕТ ЗА ОКЕАНОЛОГИЯ „NEMO“*

## 1. Софтуерен Пакет „NEMO“

*NEMO* е европейска платформа, която позволява разработката на изчисления свързани с океанографията. Тя включва серия от собствени компоненти даващи числени решения за динамиката на океаните, за ледниците (sea-ice модел), за биогеохимията, както и инстурменти за преди- и след-процесната обработка, интерфейси на другите компоненти на Земната Система (Earth System), потребителски интерфейс, компютърно зависими функции и документация на системата.

*NEMO* позволява няколко компонента на земната система да работят съвместно или разделено. Също така дава възможност за двупосочно вграждане чрез AGRIF софтуера. Свързва се с останалите елементи на земната система чрез OASIS coupler . Последната версия на NEMO включва четири компонента:

- OPA - свързан с циркулацията на океаните
- LIM2 - свързан с плътността на ледниците и термодинамиката
- LIM3 - най - съвременната версия на LIM
- TOP2 - свързан с биогеохимията

Всички компоненти могат да работят самостоятелно с изключение на LIM. NEMO е предвиден да бъде преносима платформа, която съдържа:

- Сорс кодове: основен океански циркулационен модел (OPA_SRC), допирателен линеен и помощен модел (TAM_SRC), он/оф-лайн океански tracer и био-химични модели (TOP_SRC) и „sea-ice модел“ за ледниците (LIM_SRC).
- Вграден интерфейс към OASIS coupler и IOIPSL библиотека.
- Скриптове за компилация, създаване на изпълними файлове и за изпълнение на експерименти на дадена платформа.
- Пред- и след- процесни инструменти направени на IDL (SAXO) за конфигуриране на входни и анализиране на изходни файлове.
- Стандартни конфигурации, включително три-полярен глобален океан (ORCA2). Те са предоставени с илюстративни цели позволяващи проверка на кода.
- Конфигурационна система за контрол базирана на SVN (Subversion).
- Документация за формулировката и кода на модела.

За момента са достъпни шест конфигурации, които могат да бъдат използвани, след като се свалят необходимите входни файлове:

- ***ORCA2_LIM:*** съчетана от океанска и sea-ice конфигурация базирана на ORCA триполярна решетка с 2º хоризонтална резолюция с климатологично принуждаване (climatological forcing).
- ***ORCA2_LIM_PISCES:*** като горната конфигурация с PISCES биогеохимичски модел.
- ***ORCA2_OFF_PISCES***: PISCES модел с наложен изход от ORCA2 симулация.
- ***GYRE:*** идеализирана двойно спираловидна конфигурация върху β-равнина с 1º хоризонтална резолюция с аналитично принуждаване (analytical forcing).
- ***GYRE_LOBSTER:*** като горната конфигурация с LOBSTER биогеохимически модел.
- ***POMME:*** конфигурация без граници (Open Boundary Configuration) върху експерименталната зона POMME.

Тези конфигурации непрекъснато се развиват. Целите са две: да позволяват изпълнението на конфигурации върху платформа за да се установи дали работят коректно и да дадат основа на потребителя за създаване на своя конфигурация.

## 2. Необходими софтуер и библиотеки за инсталация на „NEMO“

Преди да се инсталира софтуерния пакет „*NEMO“* е необходимо да се осигури:

- BASH
- PERL
- SVN
- Fortran90 компилатор
- netCDF (с/без hdf5 поддръжка)
- HDF5

***Bash (Bourne Again Shell)*** е безплатен команден интерпретатор (UNIX обвивка), заместител на Bourne shell (sh). Широко разпространен е за GNU операционната система и е главна обвивка за Linux и Mac OS X. Представлява текстови прозорец, позволяващ на потребителя да пише команди, които извършват действие. „*Bash*“ също може да чете команди от файл, наречен скрипт. Идентификаторите, синтаксиса и други основни свойства на езика са взети от sh, csh и ksh. *Bash* е POSIX обвивка, но с няколко разширения.

**Bash трябва да е предварително инсталиран!**

***Perl*** е език от високо ниво. Първоначално замислен да бъде скриптов език за UNIX, но в последствие претърпява множество промени. Последната версия е *Perl 6* и има много от функционалностите на други езици включващи *C, sh, AWK u sed*. Езикът се ползва за CGI скриптов език, графично програмиране, системно администриране, мрежово програмиране, финанси, биоинформатика, и други приложения.

**Perl трябва да е предварително инсталиран!**

***SVN (Apache Subversion)*** е система за контрол на версиите, разпространявана свободно. Разработчиците използват тази система за да поддържат сегашната и предходните версии на файлове като сорс кодове, уеб страници и документации.

**SVN трябва да е предварително инсталиран!**

**Fortran90 компилатор - предварително инсталиран!**

***NetCDF (Network Common Data Form)*** е набор от софтуерни библиотеки и самоописващи, машинно-независими формати за данни, които поддържат създаването, достъпа, и споделянето на масиви от научни данни. Безплатно разпространяван, поддържащ библиотеки на C, Fortran, C++, Java и други езици. От версия 4.0 нагоре, NetCDF позволява HDF5 поддръжка.

***HDF5*** е модел, библиотека и файлов формат за съхраняване и управление на данни. Поддържа голямо многообразие от типове данни и е проектиран да бъде гъвкав и ефективен вход/изход, и за обемни и комплексни данни. *HDF5* е портативен и разширяем, позволявайки на приложенията да се развиват в неговата употреба. *HDF5* технологията

включва инструменти и приложения за управление, манипулиране, разглеждане, и анализиране на данни в *HDF5 формат*. Позволява поддръжката на *zlib*.

***zlib*** е библиотека за компресиране, съдържаща методи за компресия и декомпресия, проверка за целостта на декомпресираните данни. Най-новата версия (1.2.7) използва само един метод за компресия (deflation). Компресията може да бъде направена в една стъпка, при достатъчно голям буфер или чрез няколкократно извикване на компресиращата функция. Във втория случай, приложението трябва да предостави повече входни данни и да използва изходните данни (да осигури повече място за изхода) преди всяко извикване. Компресираният формат данни използва по подразбиране *zlib*, но библиотеката позволява четенето и писането в *gzip (.gz)* формат с интерфейс подобен на *stdio* използващ функции започващи с *„gz”*.

***gzip*** форматът е различен от този на zlib. zlib форматът е проектиран да бъде компактен и бърз при използване на памет и комуникационни канали. gzip форматът е проектиран за компресирането на един файл, има по-голям хедър от zlib за да поддържа информацията на директориите и използва различен, по-бавен метод за проверка от zlib. Декодерът проверява консистенцията на компресираните данни, така че библиотеката не би трябвало да се наруши, дори и при повредени входни данни.

## 3. Инсталиране върху хетерогенен компютърен клъстер

### 3.1. Спецификация на клъстера

Хардуерната платформа е изградена на базата на многоядрени процесори и представлява специализиран компютър, индустриален стандарт за вграждане в rack 19”, процесори AMD Opteron 1216 HE Dual Core, 1MB L2 Cache, Socket AM2, 68 W, дънна платка Core Logic nVidia nForce 570 SLI MCP, Intel 6702 PXH, Form Factor 12" x 9.6", dual channel ECC memory architecture, max. 8GB DDR11800, Integrated VGA, 32MB DDR SDRAM, Single channels bus master IDE - UltraATA 100/133, 3 channel 6 SATA2 ports with RAID 0, 1, 10, 5, JBOD, 2 PCI-E Gigabit LAN, памет: 2 x 1 GB DDR-2 DIMM 800 Mhz, PC6200, твърд диск: 2 x 160 GB, оптично у-во: DVD-ROM, флопи диск: FDD 1.44 MB, кутия: 2U 19” rack mounting, захранване: 300 W. Монитор: 22” TFT WXGA+, контраст 700:1, ъгъл на виждане: хоризонтално 150°, вертикално 140°, време на реакция 5 ms, яркост 280 cd/m2, RGB + DVI-D. Допълнителни възли: базирани на процесор Intel Xeon, със следната конфигурация: процесор 2 x Quad-Core

E5310 Intel Xeon Processor 1.6GHz с EM64T, Общо: 2, FSB: 1066, L2 Кеш: 2 x 4096KB, комплект чипове (Chipset): Intel 5000P, системна памет: 4GB с възможност за разширение до 49152MB; PC2-5300 667MHz, DDR2 SDRAM; 12 DIMM слота (6 налични), активна памет: огледални памети, резервна памет, поддръжка на технология за откриване и коригиране на грешки Chipkill, BIOS: BIOS Flash ROM, контролер на диска: AIC-9580W / осем порта SAS or SATA (300MB/sec SAS) (150MB/sec SATA) / hot-swap / on planar / PCI-Express, RAID 0, 1, 10, канали: 2, конектор: вътрешна връзка с DASD backplane / един външен конектор, HDD: 2 x 146.8GB 3.5in 15K HS SAS HDD in RAID 1; общо възможност за 6 HDDs (max. 2.4TB), оптически драйв: DVD ROM / CD-RW Combo, интегрирано видео: SVGA / ATI RN50(ES1000) / on planar / 16MB DDR std/max video memory, мрежов интерфейс: двоен интегриран 10Mbps, 100Mbps, 1000Mbps, слотове за разширение: общо (свободни): 3: 1(1) PCI Express x8, 2(2) PCI Express x4 Bays: 7: 6(0) Hot-Swap HDDs Bays, 2(1) 5.25"ext, I/O ports, 6 USB (Vers 2.0), сериен (9-pin), графичен (DB-15), два Етернет (RJ-45), RJ-45 (sys mgmt), управление: адаптер за дистанционно управление Slimline, други изисквания: поддръжка на стандарт Alert Standard Format (ASF), автоматично рестартиране (ASR), индикатори на диагностика, адаптер за отдалечено управление (опция), изпълнителна среда за първоначално зареждане на операционната система, система за дистанционно инсталиране, активиране при включване в локална мрежа; изисквания за сигурност: включване на захранването при администраторска парола; система за ранно предупреждение при проблем; интегриран контролер за управление на дънната платка / мониторинг на системата и средата IPMI 2.0 H8S2166; захранване: 2 x 835W резервно захранване, 6 разклонения; размери: (ширина x дълбочина x височина): 444mm x 698mm x 85mm; тегло: 29.6 kg, възможност за вграждане: Rack Mount 2U, конвертируем за tower 26D.

### 3.2. Инсталация на библиотеките

Инсталация на *zlib-1.2.7* става със следната команда:

```
./configure --prefix=/home/Students/NEMO/zlib-1.2.7-install/

make check install
```

Инсталация на *HDF5-1.8.9* става със следните команди:

```
./configure --prefix=/home/Students/NEMO/hdf5-1.8.9-install/ --
with- zlib=/home/Students/NEMO/zlib-1.2.7-install/
```

```
make check install
```

Инсталация на *netCDF-4.1.3* със следните команди:

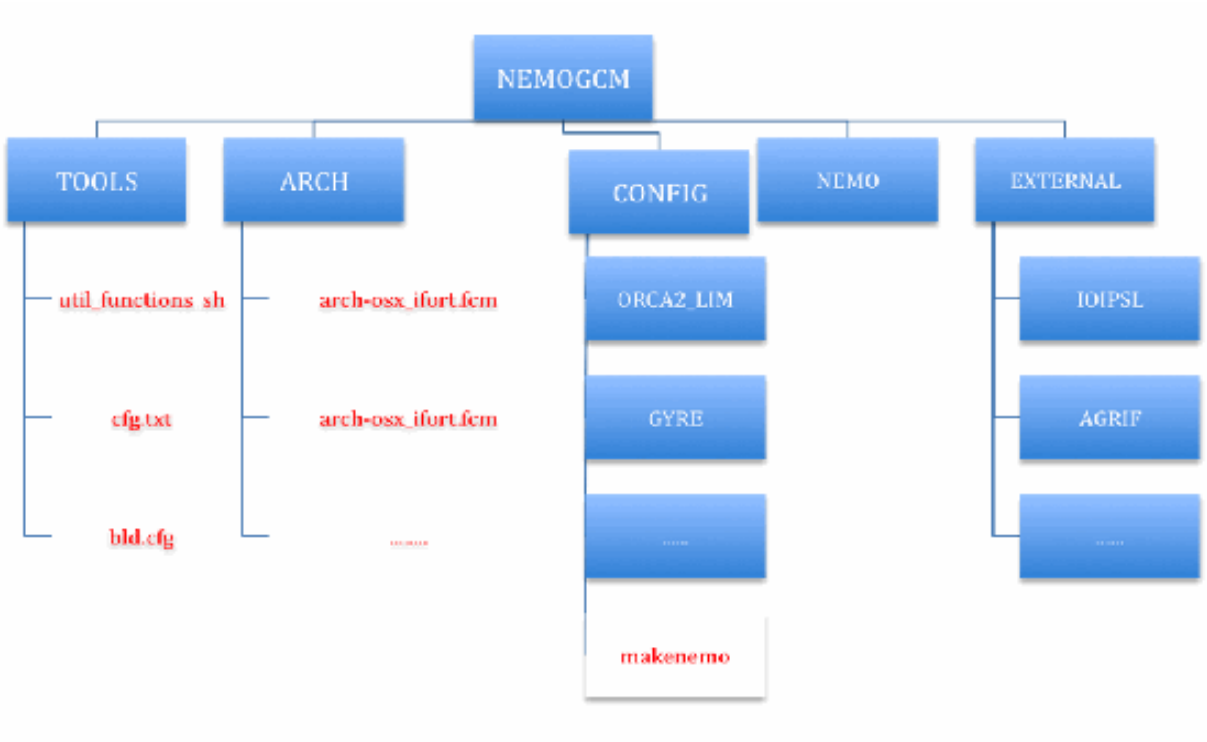
```
./configure --prefix=/home/Students/NEMO/netcdf-install/ --
enable-fortran --disable-cxx --disable-dap  CPPFLAGS='-
I/home/Students/NEMO/hdf5-1.8.9-install/include/ -
I/home/Students/NEMO/zlib-1.2.7-install/include/' LDFLAGS='-
L/home/Students/NEMO/hdf5-1.8.9-install/lib -
L/home/Students/NEMO/zlib-1.2.7-install/lib/'

make check install
```

### 3.3. Инсталиране на NEMO с FCM на компактен клъстер

**За достъп до пълната информация на официалния сайт (http://www.nemo-ocean.eu/) на софтуерния пакет, както и на достъп до самия софтуер е нужна регистрация! Тя е безплатна!**

*FCM (Flexible Configuration Manager)* се използва единствено създаването на изпълними файлове от сорс файлове.




Фиг.1. Дърво на директориите

Главният скрипт, с който се компилира и създава изпълнимия файл се нарича **makenemo** и се намира в CONFIG директорията.

В папка ARCH се намират .fcm файлове, които съдържат информация за компилаторите, неговите настройки и директории за нужните библиотеки.

Инсталацията на „NEMO“ върху компактния клъстер използва копие на *arch-gfortran_linux.fcm* файл със следното съдържание:

```
# generic gfortran compiler options for linux
# NCDF_INC    netcdf include file
# NCDF_LIB    netcdf library
# FC          Fortran compiler command
# FCFLAGS     Fortran compiler flags
# FFLAGS      Fortran 77 compiler flags
# LD          linker
# LDFLAGS     linker flags, e.g. -L<lib dir> if you have
libraries in a
# FPPFLAGS    pre-processing flags
# AR          assembler
# ARFLAGS     assembler flags
# MK          make
# USER_INC    additional include files for the compiler,  e.g.
-I<include dir>
# USER_LIB    additional libraries to pass to the linker, e.g.
-l<library>
```

---

```
%NCDF_INC            -I/home/Students/NEMO/netcdf-
install/include
%NCDF_LIB            -L/home/Students/NEMO/netcdf-install/lib -
lnetcdf -lnetcdff
%FC                gfortran
%FCFLAGS             -I/opt/bin/mpich2/include -
L/opt/bin/mpich2/lib -lmpich -lmpl -L/usr/slurm/lib -lpmi -
fdefault-real-8 -O3 -funroll-all-loops -fcray-pointer
%FFLAGS              %FCFLAGS
%LD                  gfortran
%LDFLAGS             -I/opt/bin/mpich2/include -
L/opt/bin/mpich2/lib -lmpich -lmpl -L/usr/slurm/lib -lpmi -
L/home/Students/NEMO/netcdf-install/lib
```

```
%FPPFLAGS            -I/opt/bin/mpich2/include -
L/opt/bin/mpich2/lib -lmpich -lmpl -L/usr/slurm/lib -lpmi -P -C
-traditional
%AR                  ar
%ARFLAGS             -rs
%MK                  gmake
%USER_INC            %NCDF_INC
%USER_LIB            %NCDF_LIB
```

Редовете започващи с „#” са коментари, а редовете започващи с % са променливи.

Първите две променливи съдържат include и lib директориите на NetCDF.

С FC и LD променливите указваме компилатора, а с FCFLAGS и FFLAGS опциите на компилатора.

В LDFLAGS задаваме опция на линкера, както и даваме пътя на lib директорията, която съдържа MPI библиотеките.

В FPPFLAGS добавяме include директорията съдържаща MPI хедърните файлове.

Останалото не се променя.

### 3.4. Инсталиране на NEMO с FCM на суперкомпютъра BlueGene/P

В папка ARCH се намират .fcm файлове, които съдържат информация за компилаторите, неговите настройки и директории за нужните библиотеки.

Пример: arch-ifort_linux.fcm

Неговото съдържание изглежда по следния начин:

```
# generic ifort compiler options for linux
# NCDF_INC    netcdf include file
# NCDF_LIB    netcdf library
# FC          Fortran compiler command
# FCFLAGS     Fortran compiler flags
# FFLAGS      Fortran 77 compiler flags
# LD          linker
# LDFLAGS     linker flags, e.g. -L<lib dir> if you have
libraries in a
# FPPFLAGS    pre-processing flags
# AR          assembler
```

```
# ARFLAGS     assembler flags
# MK          make
# USER_INC    additional include files for the compiler,  e.g.
-I<include dir>
# USER_LIB    additional libraries to pass to the linker, e.g.
-l<library>


%NCDF_INC            -I/usr/local/netcdf/include
%NCDF_LIB            -L /usr/local/netcdf/lib -lnetcdf
%FC                  ifort
%FCFLAGS        -r8 -O3  -traceback
%FFLAGS         -r8 -O3  -traceback
%LD                  ifort
%FPPFLAGS            -P -C -traditional
%LDFLAGS
%AR                  ar
%ARFLAGS             -r
%MK                  gmake
%USER_INC            %NCDF_INC
%USER_LIB            %NCDF_LIB
```

В следния пример е използван arch-ifort_linux.fcm файл за да се компилира GYRE конфигурация с име MY_GYRE:

```
cd NEMOGCM/CONFIG

./makenemo -m ifort_linux -r GYRE -n MY_GYRE
```

Опции за компилиране:

-n - задава се име на съществуваща или нова конфигурация

-m - име на компилатора (използва .fcm файла, но без „-arch” часта)

-h - за помощ

-r - използва конфигурация като референция за създаване на нова

-j - указва броя на процесите за компилация (при 0 няма компилация)

-d - поддиректория на NEMO (не е необходима)

-t - временна директория, която съхранява библиотеки

-clean [име_на_конфигурацията] - премахва Makefile-а и всички

създадени след това файлове. Ако не е зададена конфигурация се избира последната направена.

Пример с clean:

```
makenemo clean
```

Пример за премахване на лоша конфигурация:

```
makenemo –n ORCA2_LIM_2_2 clean_config (and answer)
```

В случая се използва новосъздадения arch-bgp.fcm (копие на arch-gfortran_linux.fcm) файл със следното съдържание:

```
# generic IBM SP
# NCDF_INC    netcdf include file
# NCDF_LIB    netcdf library
# FC          Fortran compiler command
# FCFLAGS     Fortran compiler flags
# FFLAGS      Fortran 77 compiler flags
# LD          linker
# LDFLAGS     linker flags, e.g. -L<lib dir> if you have
libraries in a
# FPPFLAGS    pre-processing flags
# AR          assembler
# ARFLAGS     assembler flags
# MK          make
# USER_INC    additional include files for the compiler,  e.g.
-I<include dir>
# USER_LIB    additional libraries to pass to the linker, e.g.
-l<library>


%NCDF_INC            -I/bgusr/ppetkov/netcdf-4.0.1-new-
bgxlf/include
%NCDF_LIB            -L/bgusr/ppetkov/netcdf-4.0.1-new-bgxlf-
dextname/lib -lnetcdf
%FC                  bgxlf90_r
%FCFLAGS             -qsuffix=f=f90 -qrealsize=8 -qnostrict -O3
-qsmp=omp -qhot -qessl -qarch=450d -qcache=auto -qtune=450 -
qextname
%FFLAGS              -qrealsize=8 -qnostrict -O3 -qsmp=omp -
qhot -qessl -qarch=450d -qcache=auto -qtune=450 -qextname
%LD                  mpixlf90_r
#bgxlf90_r
%LDFLAGS             -qextname -
L/bgsys/drivers/V1R4M2_200_2010-100508P/ppc/comm/lib
%FPPFLAGS            -P -C -I/bgsys/drivers/V1R4M2_200_2010-
100508P/ppc/comm/include
%AR                  ar
%ARFLAGS             rs
%MK                  gmake
```

```
%USER_INC            %NCDF_INC
%USER_LIB            %NCDF_LIB
```

След съхраняването на новия .fcm файл се поставя в CONFIG директорията на NEMO и може да се изпълни следната команда:

```
./makenemo -m bgp -r AMM12 -n MY_AMM12 -j 4
```

При правилно протичане на компилацията на новата AMM12 конфигурация - MY_AMM12, следва да е създаден *opa* файл в CONFIG/MY_AMM12/EXP00. Това е изпълнимият файл на конфигурацията.

Преди да се подаде като задача за изпълнение на LoadLeveler, следва да се свалят входни данни за AMM12 от официалния сайт. Те са в tar файл, съдържащ две папки и няколко gz файла. Тези две папки, както и декомпресираните gz файлове, трябва да се намират в същата директория като *opa* файла. При липса на някой от тях, след изпълнението на програмата, ще се създаде ocean.output файл, в който ще има грешка свързана с този входен файл.

За да се започне симулацията се ползва следния jcf файл с име nemo:

```
# @ job_name = NEMO AMM12 configuration tests
# @ comment = "Testing AMM12."
# @ error = err-$(jobid).err
# @ output = log-$(jobid).out
# @ environment = COPY_ALL;
# @ wall_clock_limit = 23:30:00
# @ notification = never
# @ job_type = bluegene
# @ bg_size = 128
# @ class = n0128
# @ queue
date
time  /bgsys/drivers/ppcfloor/bin/mpirun  -mode VN -np 512
./opa
date
```

Този файл се подава на LoadLeveler с командата:

```
llsubmit nemo.jcf
```

При успешно протичане на процеса, ще са се появили 5 * броя на процесите изходни nc файла, както и ocean.output, и timing.output.

Така се извършва симулацията още 4 пъти (с 640, 768, 896 и 1024 процеса) като се променят bg_size и class параметрите, съответно със стойностите 256 и n0256 за да отговарят на изискванията. Могат да се снемат следните данни:

- *Total* - общо време на процесите
- *Minimum* - минимално време за изпълнение
- *Maximum* - максимално време за изпълнение
- *Average* - средно време за изпълнение
- *Elapsed Time (s)* - изминало време в секунди
- *CPU Time (s)* - процесорно време в секунди
- *Ratio CPU/Elapsed* - съотношение между CPU Time и Elapsed Time

| | Elapsed Time (s) | CPU Time (s) | Ratio CPU/Elapsed |
|---|---|---|---|
| Total | 441513,647 | 441514,86 | 512,001 |
| Minumum | 861,161 | 861,16 | 1 |
| Maximum | 862,366 | 862,37 | 1 |
| Average | 862,331 | 862,334 | 1 |

Таблица 1: AMM12 512 процеса

| | Elapsed Time (s) | CPU Time (s) | Ratio CPU/Elapsed |
|---|---|---|---|
| Total | 280009,035 | 280010,95 | 640,004 |
| Minumum | 437,025 | 437,03 | 1 |
| Maximum | 437,544 | 437,55 | 1 |
| Average | 437,514 | 437,517 | 1 |

Таблица 2: AMM12 640 процеса

| | Elapsed Time (s) | CPU Time (s) | Ratio CPU/Elapsed |
|---|---|---|---|
| Total | 391557,172 | 391557,75 | 768,001 |
| Minumum | 509,883 | 509 | 1 |
| Maximum | 509,883 | 509,88 | 1 |
| Average | 509,84 | 509,841 | 1 |

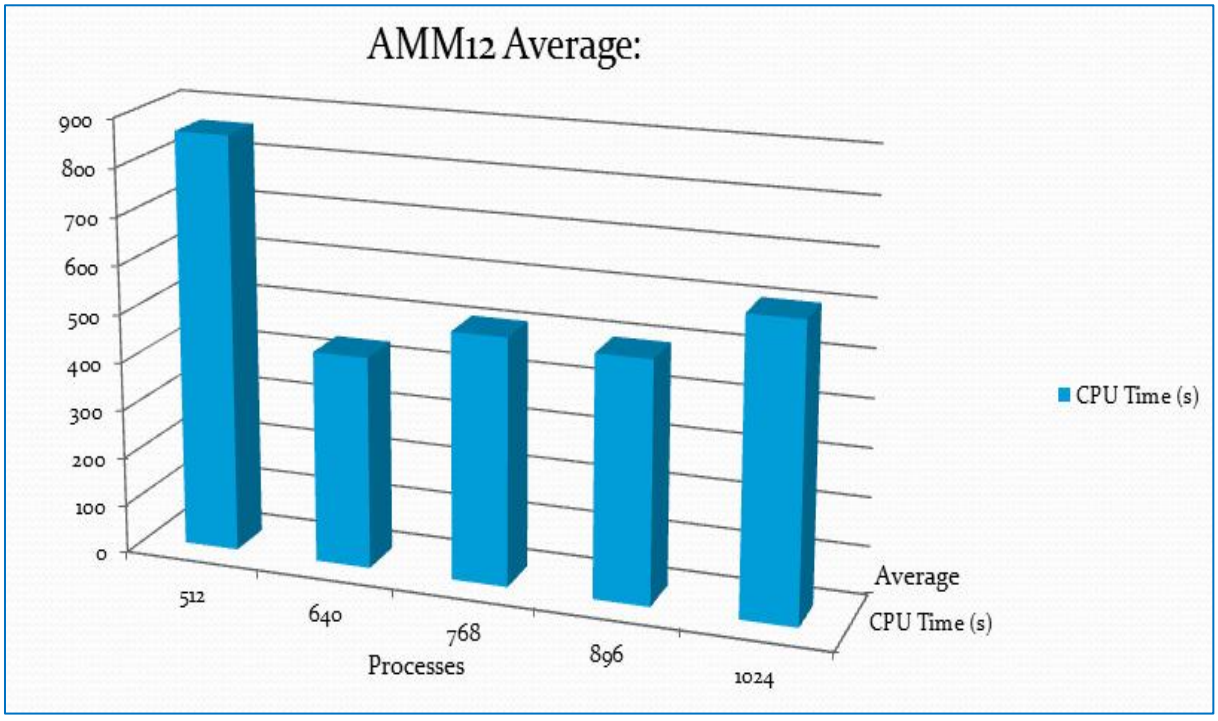
Таблица 3: AMM12 768 процеса

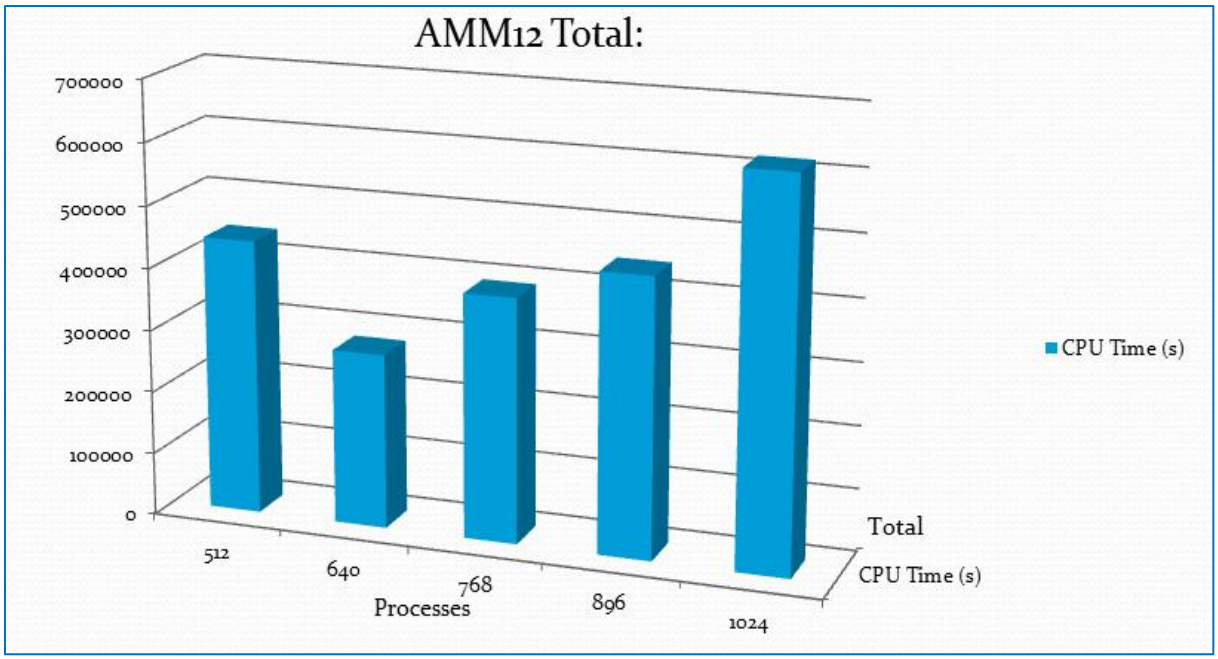
| | Elapsed Time (s) | CPU Time (s) | Ratio CPU/Elapsed |
|---|---|---|---|
| Total | 445513,579 | 445513,8 | 896 |
| Minumum | 496,126 | 496,13 | 1 |
| Maximum | 497,279 | 497,28 | 1 |
| Average | 497,225 | 497,225 | 1 |

Таблица 4: AMM12 896 процеса

| | Elapsed Time (s) | CPU Time (s) | Ratio CPU/Elapsed |
|---|---|---|---|
| Total | 619725,806 | 619724,35 | ******* |
| Minumum | 603,755 | 603,75 | 1 |
| Maximum | 605,27 | 605,27 | 1 |
| Average | 605,201 | 605,2 | 1 |

Таблица 5: AMM12 1024 процеса






## 4. Визуализация с „ParaView „

*Софтуерът „ParaView“* е с отворен код, мулти-платформено приложение за визуализация и анализ на научни бази от данни, предимно тези, които са определени в дву- или триизмерно пространство, включително тези, които се простират във времевото измерение.

Графичният потребителски интерфейс (GUI) е отворен, гъвкав и интуитивен потребителски интерфейс, който също дава фин и отворен контрол над манипулираните данни и обработка на дисплея, необходима за да се изследват и представят сложни данни.

*ParaView* има богати възможности за скриптова и групова обработка. Стандартният скриптов интерфейс използва езика за програмиране Python за скрипт контрол. С GUI, Python скрипт контрола е лесен за научаване, включително възможността за записване на действията в GUI и да ги запишете като кратки програми на Python лесни за четене.

Обработката на данни и компоненти за рендиране на ParaView са изградени върху модулна и скалируема паралелна архитектура с разпределена памет, в която много процесори работят синхронизирано в различни части на данните. Скалируемата архитектура на ParaView

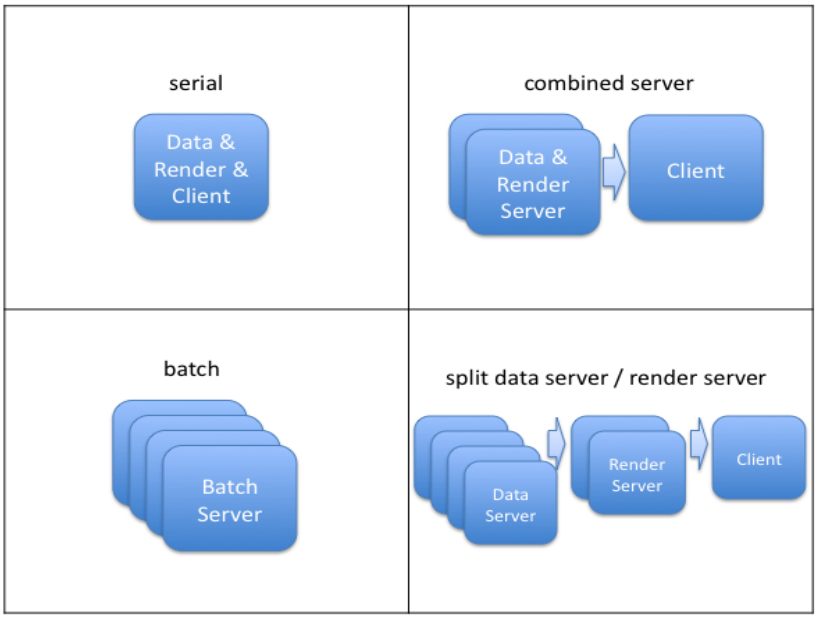
позволява да стартирате ParaView директно върху различни архитектури от малка машина като нетбук до най-големия суперкомпютър в света. Въпреки това размерът на масивите от данни, с които ParaView може да се справи на практика варира в широки граници в зависимост от размера на машината, на която се изпълняват сървърните компоненти. Поради тази причина специалистите често правят и двете, възползвайки се от архитектура клиент / сървър на ParaView.

ParaView е писано да бъде лесен за разширяване и персонализиране в нови приложения и да се използва от или да използва други инструменти. Съответно има редица различни интерфейси на ParaView за обработка на данни и визуализация, например уеб-базирания ParaViewWeb.

## Паралелна структура

*ParaView* има три основни логически компонента: клиент, сървър на данни и рендериращ сървър. Клиентът е отговорен за GUI и е интерфейс между потребителя и ParaView като цяло. Сървър на данни чете файловете и обработва данните през конвейер. Рендериращият сървър взима обработените данни и ги рендерира за да представи резултатите на вас.

Трите логически компонента могат да се комбинират в различни конфигурации. Когато се стартира ParaView, клиентът е свързан с така наречения вграден сървър, в този случай, всички три компонента съществува в рамките на един и същи процес. Освен това, можете да стартирате сървъра като самостоятелна програма и да се свържете с отдалечения клиент, или да стартирате сървъра като самостоятелна групова паралелна програма без GUI. В този случай процесът, сървърът съдържа както сървър на данни така и рендер сървър като компоненти. Сървърът може да бъде стартиран и като две отделни програми: един за сървър на данни и един за рендер сървър. Сървърни програми са паралелни програми, които могат да работят като съвкупност от независими процесите, протичащи на различни процесори. Процесите използват MPI за да координират дейността си, тъй като всеки работи над различни части от данните.




## Клиента

Клиентът е отговорен за потребителския интерфейс на приложението. Клиента на ParaView с общо предназначение е написан, за да даде възможност за мощна визуализация и анализ чрез лесен за използване интерфейс. Компонент на клиента е серийна програма, която контролира сървърните компоненти чрез API Server Manager.

## Сървър на данни

Сървъра на данни е изграден предимно от VTK четци, източници и филтри. Той е отговорен за четенето и генерирането на данни, обработката им и производство на геометрични модели, които рендер сървъра и клиента ще показват. Сървър на данни използва паралелизъм на данни чрез разделяне на данните, добавяне на призрачни нива около дяловете, както е необходимо и изпълнява синхронни паралелни филтри. Всеки процес на сървъра на данни има идентичен VTK конвейер и на всеки процес е указано кой дял на данните трябва да зарежда и обработва. Чрез разделянето на данните, ParaView е в състояние да използва цялата системна памет и по този начин е възможна обработката на големи масиви от данни.

## Рендер сървър

Рендер сървърът е отговорен за рендериране на геометрията. Подобно на сървър на данни, рендер сървъра може да се използва паралелно и има еднакви визуализационни конвейери (само за рендериране част на Конвейер). Имайки възможности за стартиране рендер сървъра отделно от сървър на данни се позволява оптималното разделение на труда между компютърни платформи. Най-големите изчислителни клъстери се използва предимно за пакетни симулации и не разполагат с хардуерен ресурс за рендериране. Тъй като не е желателно да се придвижат големи файлове с данни до отделни системи за визуализация сървъра на данни може да използва на същия клъстер, който изпълнява оригиналната симулация. Рендер сървъра може да се изпълнява на отделен клъстер за визуализация, който разполага с хардуерен ресурс за рендериране.

Възможно е да стартирате рендер сървъра с по-малко процеси, отколкото сървър на данни, но никога повече. Визуализационните клъстери обикновено имат по-малко възли от клъстерите за партидна симулация и обработваната геометрия обикновено е значително по-малка от оригиналните входни данни на симулация. ParaView събира отделните партиди на геометричния модел на сървър на данни, преди те да бъдат изпратени към сървъра за рендериране.

## Изгледи от визуализацията на океаните